Multi Table Insert

- 다중 테이블 입력?
  - 하나의 쿼리에서 액세스한 로우를 여러 개의 테이블에 동시적으로 입력
  - 과거의 여러 테이블에 입력하기 위해 쿼리를 분리하거나 다중처리(Array Processing)의 불편 해소

```
INSERT ALL
  WHEN order_total < 1000000
       THEN INTO small_orders
  WHEN order_total > 1000000 AND order_total < 2000000
       THEN INTO medium_orders
  WHEN order_total > 2000000
       THEN INTO large_orders
SELECT order_id, order_total, sales_rep_id, customer_id
  FROM orders;
```

```
Execution Plan
----------------------------------------------------------
INSERT STATEMENT
 MULTI-TABLE INSERT
   INTO OF 'SMALL_ORDERS'
   INTO OF 'MEDIUM_ORDERS'
   INTO OF 'LARGE_ORDERS'
    TABLE ACCESS (FULL) OF 'ORDERS'
```

한번 액세스한 결과를 여러 테이블에 제공

Multi Table Insert

파일시스템으로 부터 관계형 데이터베이스 구조로의 이행할 때 효과적임(External Table도 활용가능)

```
INSERT ALL
  WHEN ottl < 100000
    THEN INTO small_orders VALUES(oid, ottl, sid, cid)
  WHEN ottl > 100000 and ottl < 200000
    THEN INTO medium_orders VALUES(oid, ottl, sid, cid)
  WHEN ottl > 200000
    THEN into large_orders VALUES(oid, ottl, sid, cid)
  WHEN ottl > 290000
    THEN INTO special_orders
SELECT o.order_id oid, o.customer_id cid,
          o.order_total ottl, o.sales_rep_id sid, c.SAL cl, c.job cem
   FROM  orders o, emp c
 WHERE o.customer_id = c.empno
     AND o.order_date between :b1 and :b2;
```

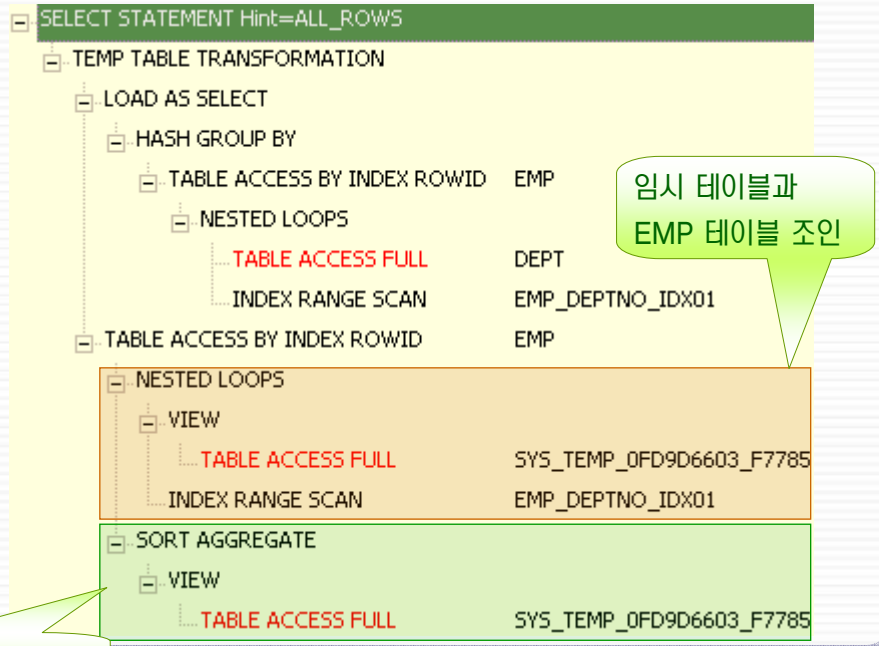
```
Execution Plan
----------------------------------------------------
INSERT STATEMENT
 MULTI-TABLE INSERT
  INTO OF 'SMALL_ORDERS'
  INTO OF 'MEDIUM_ORDERS'
  INTO OF 'LARGE_ORDERS'
  INTO OF 'SPECIAL_ORDERS'
  HASH JOIN
   TABLE ACCESS (FULL) OF 'EMP'
   TABLE ACCESS (BY INDEX ROWID) OF 'ORDERS'
    INDEX (RANGE SCAN) OF 'ORDERS_IDX2'(NON-UNIQUE)
```

다양한 방법의 조인이 가능

서브쿼리 팩토링 (With 절)

- 서브쿼리 팩토링?
  - 복잡한 쿼리문에 대해 WITH절을 사용하여 생성한 결과를 임시 테이블에 저장
  - 테이블과 동일하게 쿼리 문에서 사용가능
  - 개념적으로는 인라인 뷰와 거의 동일
- 인라인 뷰 VS 서브쿼리 팩토링
  - 인라인 뷰는 건건이 실행, 서브쿼리 팩토링은 단 한번만 실행
  - 한 번 복잡한 가공을 한 결과를 하나의 쿼리에서 여러 번 사용해야 할 때 효율적임

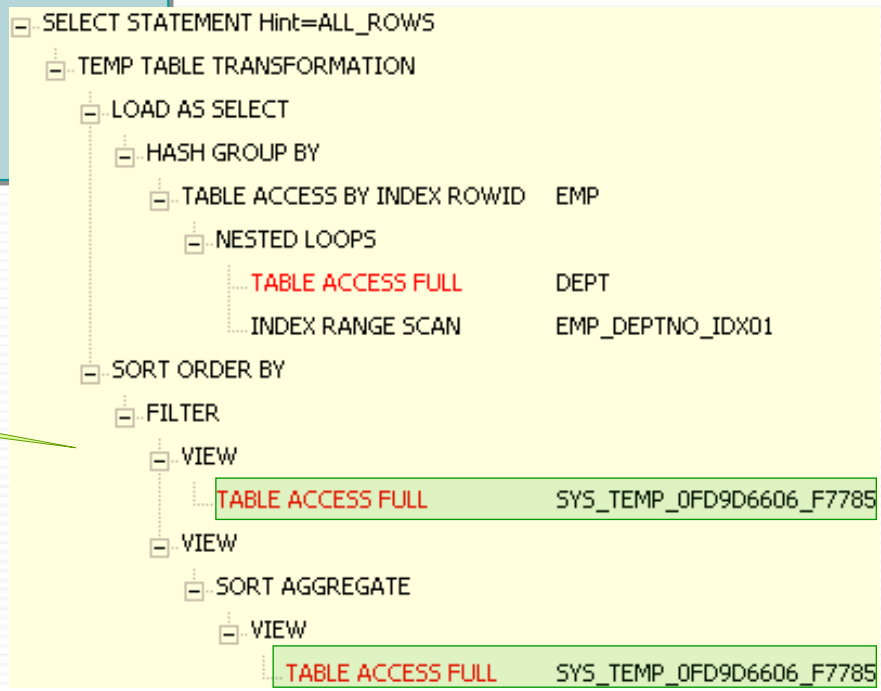
```
WITH total_sal AS
    (SELECT  D.deptno, D.loc, E.job, sum(E.sal) tot_sal
       FROM  emp E, dept D
      WHERE  E.deptno = D.deptno AND E.hiredate > :b1
      GROUP BY D.deptno, D.loc, E.job )
SELECT e.empno, e.ename, e.sal, e.sal/t.tot_sal sal_percent
  FROM  emp e, total_sal t
 WHERE e.deptno = t.deptno
     AND e.sal > (SELECT max(tot_sal)
                    FROM total_sal
                   WHERE job = 'CLERK');
```



서브쿼리 팩토링 (With 문)

```
WITH
  dept_costs AS
  ( SELECT dname, SUM(sal) dept_total FROM  emp e, dept d
    WHERE e.deptno = d.deptno
    GROUP BY dname),
  avg_cost AS
  (SELECT SUM(dept_total)/COUNT(*) avg_sal FROM dept_costs)
  SELECT *
    FROM  dept_costs
 WHERE dept_total > (SELECT avg_sal FROM avg_cost)
  ORDER BY dname;
```

동일한 임시 테이블을 사용하고 있음



Merge 문

처리대상 테이블

정보 제공 집합

```
MERGE INTO bonuses D
USING (SELECT employee_id, salary, department_id
         FROM  employees
        WHERE department_id = 80) S
ON (D.employee_id = S.employee_id)
WHEN MATCHED       THEN UPDATE SET D.bonus = D.bonus + S.salary*.01
                        DELETE WHERE (S.salary > 8000)
WHEN NOT MATCHED THEN INSERT (D.employee_id, D.bonus)
                        VALUES (S.employee_id, S.salary*0.1);
```

연결고리

존재 시 처리

부재 시 처리

- 처리 대상에 갱신, 삭제, 입력 등이 동시 발생
  - 하나의 SQL에서 처리가 불가능
  - PL/SQL을 이용한 처리
  - ARRAY PROCESSING을 이용한 처리
- MERGE 문을 활용한 처리 방안

```
Execution Plan
----------------------------------------------------
MERGE STATEMENT
  MERGE OF 'BONUSES'
    VIEW
      NESTED LOOPS (OUTER)
        TABLE ACCESS (BY INDEX ROWID) OF 'EMPLOYEES'
          INDEX (RANGE SCAN) OF 'EMPLOYEE_X1' (NON-UNIQUE)
        TABLE ACCESS (BY INDEX ROWID) OF 'BONUSES'
          INDEX (RANGE SCAN) OF 'BONUSES_X1' (NON-UNIQUE)
```

먼저 GROUP BY 집합을 생성해 둠

정보 제공 집합이 기준이 됨